組込みソフトウェアにおける 耐障害ソフトウェアアーキテクチャの信頼性評価

鈴木有也[†] 守谷友和[††] 後藤英樹[††]

N バージョンプログラミング（NVP），アクセプタンスボーティング（AcV）等に代表される耐障害ソフトウェアアーキテクチャ（以下，耐障害アーキテクチャ）は，組込みソフトウェアの異常検出・回復の一手段として知られている．しかしながら，実際のソフトウェアに対して耐障害アーキテクチャを適用した場合の定量的な信頼性評価は，NVP の評価が多数で，それ以外の耐障害アーキテクチャに関しては報告されてこなかった．そこで，本研究では，NVP 以外の耐障害アーキテクチャに対する定量的な信頼性評価を目的とする．例として，回復に必要な最小バリアント数が 2 個で実装コストを抑えられる AcV を車載制御ソフトウェアに対して適用し，異常検出・回復性能を評価する．結果として，異常検出時の AcV の回復率が NVP を上回ることが示され，AcV が，NVP と比較して車載制御ソフトウェアの耐障害化に有効であることがわかった．

Reliability Evaluation of Fault-Tolerant Software Architecture for Embedded Software

Naoya Suzuki,[†] Tomokazu Moriya[††] and Hideki Goto[††]

N-version programming(NVP), acceptance voting(AcV), etc. are popular fault-tolerant software architectures(FTSAs) that detect and recover the failure of embedded software. However except NVP, there are few reports about quantifying the reliability of practical embedded software to which FTSA is applied. We quantify and evaluate the reliability of the software to which FTSA is applied except NVP. We focus on AcV that requires only two variants to recover and can reduce implementation costs, apply it to in-vehicle control software, and evaluate the reliability by quantifying failure detection and recovery performance. As a result the recovery performance of AcV is superior to that of NVP. We conclude that AcV is more effective than NVP in in-vehicle control software.

1. はじめに

近年，組込みソフトウェアの大規模化・複雑化が顕著に進んでいる[12]．組込みソフトウェアの信頼性を将来にわたって維持していくための一手段として挙げられるのが，機器動作時における組込みソフトウェアの異常検出・回復である．ここでいう異常というのは，出荷後の組込みソフトウェアに存在するバグが原因で，設計者の意図通りに組込みソフトウェアが動作しない状態を指す．そのような異常を検出し回復する技術としては，N バージョンプログラミング（NVP），アクセプタンスボーティング（AcV），リカバリブロック等に代表される耐障害ソフトウェアアーキテクチャ（以下，耐障害アーキテクチャと略す）が知られている[8)10)]．

耐障害アーキテクチャの特徴は，ソフトウェア冗長化，時間冗長化，またはデータ冗長化に基づくソフトウェア設計を施すことによって，ソフトウェアの異常検出と回復を行えることである．それにより，仕様に沿った正しい出力値が得られる可能性を高めることができる．しかしながら，実際の組込みソフトウェアに耐障害アーキテクチャを適用した場合の定量的な信頼性評価は，NVP が多数で，それ以外の耐障害アーキテクチャに関しては報告されてこなかった．例えば，Cai ら[3)] は，香港大において 34 チームを組織し，航空宇宙系のクリティカルモジュールを対象に 34 個のバリアント（同一の仕様から作成する冗長化プログラム）を作成し，それらを用いて NVP の信頼性を定量的に評価している．また，Townend ら[9)] は，工場のセル生産方式の制御モジュールに対して NVP を適用

† 株式会社豊田中央研究所
Toyota Central R&D Labs., Inc.
†† トヨタ自動車株式会社
Toyota Motor Corporation

し，シングルプログラム（冗長化する前のプログラム）と比較した定量的な信頼性評価を行っている．これらの先行研究のように，耐障害アーキテクチャの最も一般的な形態である NVP が信頼性評価の対象として選ばれる傾向にある．

一方，組込みソフトウェアへの耐障害アーキテクチャの適用を考えた場合，NVP が常に耐障害アーキテクチャとして最適とは限らない．なぜならば，耐障害アーキテクチャを適用する組込みソフトウェアの要件は多種多様で，その要件を満たす可能性の高い耐障害アーキテクチャを選択するのが妥当だからである．例えば，実行時間のデッドラインに余裕があり，かつバリアントの実装コストを抑える必要がある場合は，複数のバリアントを必要せず，時間冗長化を用いたリトライで回復可能なチェックポインティング＆ロールバックを用いることができる．しかしながら，NVP 以外の耐障害アーキテクチャが適用に妥当であると判断された場合，その耐障害アーキテクチャを適用することでどれほどの信頼性が得られるのかが定量的に明らかになっていないのが現状である．

そこで，本研究では，NVP 以外の耐障害アーキテクチャを実際の組込みソフトウェアに適用した場合の定量的な信頼性評価を目的とする．組込みソフトウェアの例として車載制御ソフトウェアの通信層（以下，車載通信層と略す）を取り上げ，それに適する耐障害アーキテクチャを定性的な観点から選択する．その後，選択した耐障害アーキテクチャを車載通信層に適用し，評価方法にバグインジェクションを用いて信頼性を定量的に評価する．その評価結果を，NVP を適用した場合と比較することで，車載通信層の信頼性向上に一層効果のある耐障害アーキテクチャを定量的な観点から裏付けることができる．

本稿の構成は，以下のとおりである．第 2 章で，車載通信層に適する耐障害アーキテクチャの選択について述べる．第 3 章で耐障害アーキテクチャの適用と評価について説明し，第 4 章で評価結果を示す．第 5 章で本稿をまとめる．

2. 耐障害アーキテクチャの選択

耐障害アーキテクチャは，いくつかの代表的な方式に分類され，それぞれにメリットとデメリットがある[8]．よって，耐障害アーキテクチャを組込みソフトウェアに適用する際には，耐障害アーキテクチャの候補の中から，適用対象の組込みソフトウェアに最適と思われる特性（対応できる異常の種類，リアルタイム性，実装コスト等）を有する耐障害アーキテクチャを選択する必要がある．本章では，耐障害アーキテクチャの代表的な方式とそれぞれの特性について述べた後，耐障害アーキテクチャを適用する組込みソフトウェアの例として車載通信層を取り上げ，最適な耐障害アーキテクチャを，文献 8) で列挙されている中から選択する．

2.1 耐障害アーキテクチャの方式と特性

（1）冗長化方式：主にソフトウェア冗長化，データ冗長化，および時間冗長化の 3 つの方式が挙げられる．1 つ目のソフトウェア冗長化は，ある単一の仕様から，ソフトウェア全体，またはその一部を複数作成する冗長化手法である．それら冗長化したソフトウェアをバリアントと呼ぶ．ソフトウェア冗長化の目的は，ソフトウェアに多様性を持たせることにより，ある入力で複数のバリアントに同時に異常が発生する可能性を低減することにある．これにより，同時に異常が発生することさえなければ，多様な原因で発生する幅広い異常に対応可能である．耐障害アーキテクチャの候補例としては，NVP，AcV 等が挙げられる．

2 つ目のデータ冗長化は，入力データに多様性を持たせる冗長化手法である．その目的は，本来の入力値と比較してわずかに異なる入力を与えることによって，異常が発生する入力列を回避することにある．耐障害アーキテクチャの候補例としては，リトライブロック，N コピープログラミング等が挙げられる．

3 つ目の時間冗長化は，時間というリソースを余分に用いて，異常発生時に当該ソフトウェアを再度実行させる冗長化手法である．その目的は，異常が発生する一時的な環境（例えば変数への特定のアクセスタイミング等）を回避することにある．耐障害アーキテクチャの候補例としては，チェックポインティング＆ロールバック，リカバリブロック等が挙げられる．

（2）回復方式：大別してフォワードリカバリとバックワードリカバリの 2 つの方式が挙げられる．1 つ目のフォワードリカバリは，複数あるバリアントの結果から，異常が発生していないバリアントの計算結果を出力とすることで処理を回復する方法である．この方式は，回復時にリトライを必要としないために，回復時の時間的な遅延を最小限に抑えることができるという利点があり，それゆえリアルタイムアプリケーションに有効である．耐障害アーキテクチャの候補例としては，NVP，AcV，N コピープログラミング等が挙げられる．

2 つ目のバックワードリカバリは，時間冗長化を前提としたうえで，リトライを行って正常動作に戻す方法である．この方式の利点は，バリアントが必ずしも複数必要でないため，設計実装コストの増加を抑えら

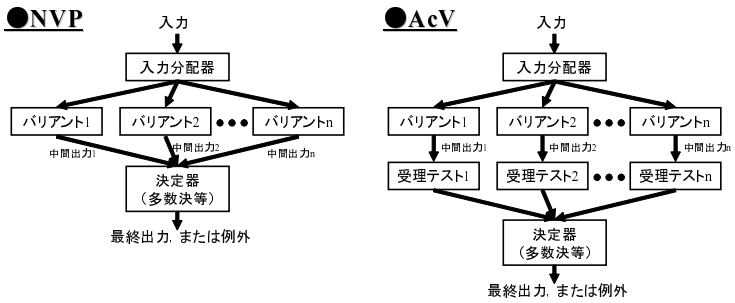

図 1 NVP と AcV のアーキテクチャ
Fig. 1 The architecture of NVP and AcV

れることが挙げられる．一方，欠点としては，リトライによって実行時間の増加が避けられないため，リアルタイム性の確保が困難なことが挙げられる．耐障害アーキテクチャの候補例としては，リカバリブロック，リトライブロック等が挙げられる．

2.2 評価適用事例

本研究では，耐障害アーキテクチャを適用する組込みソフトウェアの一例として，CAN 通信を実現する車載通信層を取り上げる．車載通信層は，車載 LAN で接続される大多数の電子制御ユニット（Electronic Control Unit，以下，ECU と略す）に搭載される．そのうえ，他の ECU と連携しながらサービスを提供する車載制御アプリケーションを動作させるうえで，通信動作の継続は必要不可欠である．車載通信層は，耐障害アーキテクチャの適用例として好適といえる．

車載通信層に適用する耐障害アーキテクチャが満たすべき要件としては，第 1 に幅広い異常への対応，第 2 にリアルタイム性の確保，および第 3 に実装コストの低減の 3 点が挙げられる．第 1 の要件である幅広い異常への対応には，冗長化方式にソフトウェア冗長化を用いることが考えられる．また，第 2 の要件であるリアルタイム性の確保には，回復方式にフォワードリカバリを用いるとよい．よって，ソフトウェア冗長化方式を用い，かつフォワードリカバリ方式の耐障害アーキテクチャ（NVP，AcV，コンセンサスリカバリブロック等）が適用候補となる．さらに，第 3 の要件である実装コストの低減を考えると，これらの候補の中でも，第 2.3 節に示すように，回復に必要な最小バリアント数が 2 個である AcV が最も効果が期待できる選択と考えられる．

2.3 AcV の特徴

n 個のバリアントから構成される AcV のアーキテクチャを，NVP とともに**図 1** に示す．AcV の特徴は，各バリアントで得られた出力（以降，中間出力と呼ぶ）を，受理テストでチェックし，それをパスした中間出力のみ決定器に入力することである．受理テストとは，バリアントの出力が正しいか否かを判定する機構である．例えば，アサーション，チェックサム等で実装され，中間出力が正常であることの必要条件をパスするか否かをチェックする機構であることが多い．よって，受理テストをパスしない場合は，中間出力が明らかに異常であることを示すため，決定器へ入力せず破棄する．決定器では，入力された中間出力に対して決定アルゴリズム（多数決等）を適用し，最終的な出力（以降，最終出力と呼ぶ）を決定する．もし，決定器に入力される中間出力が 1 個の場合は，決定器は，その中間出力を無条件で最終出力とする．決定器が最終出力を決定できない場合（例えば多数決が取れない場合等）は，異常を検出したものの回復できないとして例外を発行する．

AcV の第 1 の利点は，誤った中間出力を正確に破棄することができる厳密な受理テストを用意できるなら，バリアントに共通するバグによって発生した異常を除去できる可能性が高くなるため，NVP よりも高い検出回復性能が得られる可能性があることである．また，第 2 の利点は，回復に必要な最小バリアント数が，NVP では 3 個であるのに対し，AcV では 2 個であることである．AcV では，バリアント数が 2 個の場合，たとえ片方のバリアントが受理テストをパスしなかったとしても，もう片方のバリアントの中間出力が受理テストをパスすれば，決定器が，受理テストをパスした中間出力を最終出力として採用できるため，回復可能である．

3. 耐障害アーキテクチャの適用と評価

本章では，本評価で用いる評価方法であるバグインジェクションについて説明し，バグインジェクションをする評価対象とそれらに対する評価項目について述

表 1 バグインジェクションする評価対象
Table 1 The evaluation items for bug injection

バグインジェクション数☆	バリアント数 n	耐障害アーキテクチャ	インジェクションするバグの種類☆☆
1	1	シングル	バグ A, バグ B, バグ C
	2	NVP	バグ A-バグ B, バグ A-バグ C, バグ B-バグ C
		AcV（AT 強／ AT 弱）	バグ A-バグ B, バグ A-バグ C, バグ B-バグ C
	3	NVP	バグ A-バグ B-バグ C
		AcV（AT 強／ AT 弱）	バグ A-バグ B-バグ C
2	1	シングル	バグ A+B, バグ A+C, バグ B+C
	3	NVP	バグ A+B-バグ A+C-バグ B+C
		AcV（AT 強／ AT 弱）	バグ A+B-バグ A+C-バグ B+C

べる．その後，バグ顕在化シナリオについて説明し，評価手順を示す．

3.1 評価方法

耐障害アーキテクチャを適用した組込みソフトウェアに対して，適用した耐障害アーキテクチャが妥当であるかどうかを評価する一手段として，バグインジェクションが挙げられる[4)7)]．バグインジェクションによる評価では，組込みソフトウェアに対してバグを注入（インジェクション）し，そのバグが顕在化した時に，耐障害アーキテクチャが想定通り動作して，バグの顕在化を検出し回復できるかを確かめることによって，適用した耐障害アーキテクチャの妥当性を評価する．

インジェクションするバグの例としては，その組込みソフトウェアのドメインで発生する典型的なバグ事例，および世の中の一般的なバグ事例が挙げられる．特に，後者に関しては，文献 6) において，計算式の誤り，無限ループ等，100 項目以上列挙されている．インジェクションするバグは，組込み機器動作時に顕在化する可能性の高いバグであればあるほど，耐障害アーキテクチャが実際に使用される状況に一致するため，バグインジェクションの評価に適しているといえる．

インジェクションした後は，当該ソフトウェアを動作させてバグを顕在化させ，その時に耐障害アーキテクチャが想定通り動作して，異常を検出し回復できるかを評価する．バグを顕在化させるための入力時系列を，本稿ではバグ顕在化シナリオと呼ぶ．例えば，あるソフトウェアに対し，任意の時刻 t において，入力 x に 0 を与えたときにゼロ割りのバグが顕在化するならば，$\{(t,x)|(0,1),(1,3),(2,0)\}$ はバグ顕在化シナリオである．なぜなら，$(t,x)=(2,0)$ のときにゼロ割りが顕在化するからである．このようなバグ顕在化シナリオを複数用意することによって，バグのさまざまな顕在化の仕方に対して異常検出・回復が可能かどうかを評価することができる．以降，バグ顕在化シナリオを構成する各入力（上記の例では (0, 1) 等）をテストケースと呼ぶことにする．

3.2 評価対象

本評価では，耐障害アーキテクチャを適用する前の車載通信層（シングルプログラム，略してシングルと呼ぶ）と，２種類の耐障害アーキテクチャ（NVP と AcV）を適用した車載通信層のそれぞれにバグインジェクションし，信頼性を評価する．以降，説明を簡潔にするために，シングルを，耐障害アーキテクチャの種類に便宜的に含めることとし，シングルのバリアントといった場合は，シングルそのものを指すこととする．インジェクションするバグとしては，自動車向け機能安全規格 ISO26262[5)] で列挙されている一般的な通信異常のうち，(1) データ抜け，または (2) データ到着タイミングずれの 2 種類の異常を引き起こすバグをインジェクション例とする．具体的には，(1) データ抜けを引き起こすバグ事例として，(A）送信データ書き込み先ミス（送信データを受信スロットに定期的に誤って書き込む），ならびに（B）送信バッファの上書き（送信遅延発生時に送信バッファに残っている送信待ちのデータを上書きする）を例とする．また，(2) データ到着タイミングずれを引き起こすバグ事例として（C）オフセット無視（オフセット時間を待たずに通信を開始する）を例とする．本稿では，これらを，それぞれバグ A，バグ B，ならびにバグ C と呼ぶ．

バグインジェクションする評価対象を**表 1** に示す．本表は，バリアント数 n からなる各耐障害アーキテクチャに対して，個々のバリアントに対するバグインジェクション数☆と，インジェクションするバグの種類☆☆を示している．受理テストの判定性能の違いが，得られる信頼性にどのような影響を与えるのかを評価

☆ 各バリアントに対するバグインジェクション数を指す．例えば，バリアント数 n=3 の耐障害アーキテクチャでバグインジェクション数が 2 であれば，各バリアントに 2 個ずつ，計 6 個のバグがインジェクションされることを示す．

☆☆ 例えば，バリアント数 n=3 の耐障害アーキテクチャに対して，各バリアントにそれぞれ，バグ A とバグ B，バグ A とバグ C，バグ B とバグ C をインジェクションした場合，バグ A+B-バグ A+C-バグ B+C と表現する．

するために，AcV の場合は，判定性能の異なる受理テスト（AT：Acceptance Test）を 2 種類用意して評価する．すなわち，バリアントの出力の正しさを厳密に判定できる受理テストを AT 強，AT 強よりも厳密さを緩めたそれを AT 弱として，それら 2 種類の受理テストを使用した場合を評価する．具体的には，AT 弱は，通信アプリケーションからの送信データが正しく CAN レジスタに書かれたかをチェックするアサーションであり，AT 強は，それに加えてオフセットが正しいかをもチェックするアサーションである．

各耐障害アーキテクチャのバリアント数 n は，1 個，2 個，または 3 個とし，各バリアントに対してバグを 1 個，ないし 2 個インジェクションする．バグインジェクション数が 1 個の場合は，独立したバグが各バリアントに存在するケースを模擬している．この場合，ほとんどのテストケースにおいて，単独のバリアントでのみバグが顕在化し，その他のバリアントでは顕在化しない．まれに，あるテストケースにおいて，複数のバリアントで同時に異なる種類のバグが顕在化する．

一方，バグインジェクション数が 2 個の場合は，共通のバグがバリアントに存在するケースを模擬している．この場合，あるテストケースにおいて，複数のバリアントで同時に同一のバグが顕在化する．インジェクションするバグの種類では，インジェクションしうる 3 種類のバグの組合せを列挙している（NVP と AcV においてすべてのバリアントに同一のバグをインジェクションする組合せは除く）．これらの各組合せに対して，バグ顕在化シナリオを入力し，評価を行う．詳細な評価項目に関しては，第 3.3 節で述べる．

3.3 評価項目

本研究では，**表 2** に示すように，正確性と耐障害性の 2 種類の指標によって信頼性を定量化する．前者の正確性は，ISO/IEC 9126[2] で定義されている正確性と同義であり，どれだけ仕様通りに出力が得られるかを表す指標である．本研究では，正確性を正解率，例外率，および誤り率によって定量化する．これらの中で最も重要なのが正解率であって，全テストケース数のうち仕様通りの最終出力が得られた割合を指す．本稿において，正確性が高いとは，正解率が高いことを意味する．一方，例外率は，全テストケース数のうち例外を出力した割合を指す．例外率が高いということは，何らかの異常を検出したときに例外を発行することによって，誤った最終出力を出さない能力に優れていることを指す．また，誤り率は，全テストケース数のうち仕様通りか例外以外の最終出力が得られた割合を指す．誤り率が高いということは，異常が発生した場合にその異常が検出されず，そのまま最終出力として出力される割合が高いことを示す．例外率と誤り率を示すことによって，仕様通りの最終出力が得られなかった場合に，その異常を検出できたのか，それとも検出できずに出力してしまったのかを，全テストケース数に対する割合で示すことができる．

後者の耐障害性に関しては，どれだけ異常の発生を検出でき，回復できるかを表す指標であり，ISO/IEC 9126 で定義されている障害許容性と回復性に相当する．本研究では，耐障害性を異常発生数，検出数，回復数，検出率，および回復率によって定量化する．本稿において，耐障害性が高いとは，検出率と回復率がともに高いことを指す．検出率と回復率は，異常発生数，検出数，および回復数によって求められる．異常発生数とは，全テストケース数のうち，少なくとも 1 個のバリアントの中間出力が異常であった個数である．例えば，バリアント数 n=3 の耐障害アーキテクチャに対して，あるテストケースを与えたときに，3 個のうち 1 個のバリアントの中間出力に異常が発生した場合，異常発生数にカウントされるのは 1 個である．同時に 2 個のバリアントに異常が発生した場合も，異常発生数にカウントされるのは 1 個である．検出数とは，異常発生数のうち，少なくとも 1 個のバリアントの中間出力が異常であることを検出できた個数であり，回復数とは，異常発生数のうち，例外を出さずに回復できて仕様通りの最終出力が得られた個数である．これら異常発生数，検出数，および回復数を用いると，検出率と回復率が求められる．検出率とは，異常発生数のうちの検出数の割合であり，回復率とは，異常発生数のうちの回復数の割合である．これら検出率と回復率を示すことによって，耐障害性を定量化する．

3.4 バグ顕在化シナリオ

適用した耐障害アーキテクチャが異常を検出し回復できるかを評価するには，バグ A，B，および C を顕在化させるバグ顕在化シナリオ（時系列のテストケース）が必要となる．バグ A（送信データ書き込み先ミス）は，任意の定期的な送信データを与えることで顕在化できる．同様に，バグ B（送信バッファの上書き）は，送信遅延を発生させることで，バグ C は，オフセット時間を設定することで顕在化できる．よって，本研究では，インジェクションしたバグを顕在化させるための仮想のアプリケーションを想定し，以下に示すバグ顕在化シナリオを用意する．すなわち，テストケースとして送信データを 10000 個用意し，5[ms] 間隔で定期送信する．オフセット時間を 20[ms] に設定し，送信遅延の発生を 45000[ms] の時点で発生させ

表 2 信頼性の評価項目
Table 2 The evaluation items of reliability

指標	評価項目	説明
正確性	正解率	全テストケース数のうち，仕様通りの最終出力が得られた割合
	例外率	全テストケース数のうち，例外を出力した割合
	誤り率	全テストケース数のうち，仕様通りか例外以外の最終出力が得られた割合
耐障害性	異常発生数	全テストケース数のうち，少なくとも 1 個のバリアント出力が異常であった個数
	検出数	異常発生数のうち，少なくとも 1 個のバリアントの中間出力の異常を検出できた個数
	回復数	異常発生数のうち，例外を出さずに回復できて仕様通りの最終出力が得られた個数
	検出率	異常発生数のうちの検出数の割合
	回復率	異常発生数のうちの回復数の割合

る．バグ A は 500[ms] に 1 回の周期で顕在化するようにバグインジェクションする．このようなバグ顕在化シナリオにおいて，評価項目に挙げた各項目を測定する．

3.5 評 価 手 順

耐障害アーキテクチャを適用した車載通信層を，バグ顕在化シナリオに沿って動作させるためには，以下の手順を要する．

(1) バグ顕在化シナリオを構成するテストケースを，耐障害アーキテクチャ（シングル・NVP・AcV）を適用した車載通信層に渡す．
(2) 渡された各テストケースに対して，耐障害アーキテクチャを適用した車載通信層を動作させる．
(3) (2) の結果，どのようなデータが最終出力となって送信されたかをロギングする．
(4) 本来送信されるべきデータと比較して，正確性と耐障害性を導き出す表 2 の評価項目を得る．

これらの手順を実行するにあたって，本研究では，Cygwin[1] 上に車載通信層のシミュレータを構築し評価に用いる．本シミュレータは，C 言語，ならびに C++ 言語用の単体テストドライバである Cutter[11] のデータ駆動テスト機能を使用することによって，C 言語で実装された車載通信層に対してバグ顕在化シナリオを入力し，その結果をロギングできる．また，本シミュレータは，3 個までの車載通信層のバリアントを用いて，本評価対象の耐障害アーキテクチャである NVP と AcV を構築できる．

4. 評 価 結 果

本章では，第 3 章で述べた評価方法に従って，車載通信層に適用した耐障害アーキテクチャの正確性と耐障害性を評価する．正確性と耐障害性それぞれ，(1) 独立したバグが存在する場合（バグインジェクション数 1），ならびに (2) 共通のバグが存在する場合（バグインジェクション数 2）について評価結果を示す．

表 3 独立したバグが存在する場合の正確性（単位：%）
Table 3 The accuracy with independent bugs

耐障害アーキテクチャ		正解率	誤り率	例外率
n=1	シングル	96.59	3.41	0
n=2	NVP	93.19	0	6.81
	AcV, AT 弱	99.97	0	0.03
	AcV, AT 強	99.99	0	0.01
n=3	NVP	99.98	0	0.02
	AcV, AT 弱	100	0	0
	AcV, AT 強	100	0	0

4.1 正確性の評価

(1) 独立したバグが存在する場合

独立したバグが存在する場合（バグインジェクション数 1）の正確性を**表 3** に示す．表 2 で示したように，正解率，誤り率，および例外率で正確性を定量化している．バリアント数 n=1，ならびに n=2 の場合は，表 1 に示す「インジェクションするバグの種類」の 3 パターン（例．n=1 の場合はバグ A，バグ B，およびバグ C）の平均値を示している．この結果について，以下のように考察できる．

[AcV で正解率が向上]

n=1 のシングルでは，正解率が 96.59%であるのに対し，n=2 の NVP を除いた，n=2 と n=3 の項目については，99.97%〜100%とシングルを上回る正解率が得られている．AcV，または n=3 の NVP を用いることで，あるバリアントでバグが顕在化した場合でも，バグが顕在化しない他の中間出力を最終出力として採用することで正しい出力が得られるといえる．特に着目すべき点は，n=2 の AcV（AT 強）が n=3 の NVP を超える正解率を得ていることである．受理テストを厳密にすることで，バリアント数を増加するよりも高い正解率が得られる一例が示されている．

[AcV で誤り率がゼロかつ例外率が減少]

誤り率に着目すると，シングルが 3.41%であるのに対し，NVP と AcV を用いた場合はゼロである．この理由としては，シングルのバグが顕在化した場合には，異常値をそのまま出力してしまうのに対し，NVP

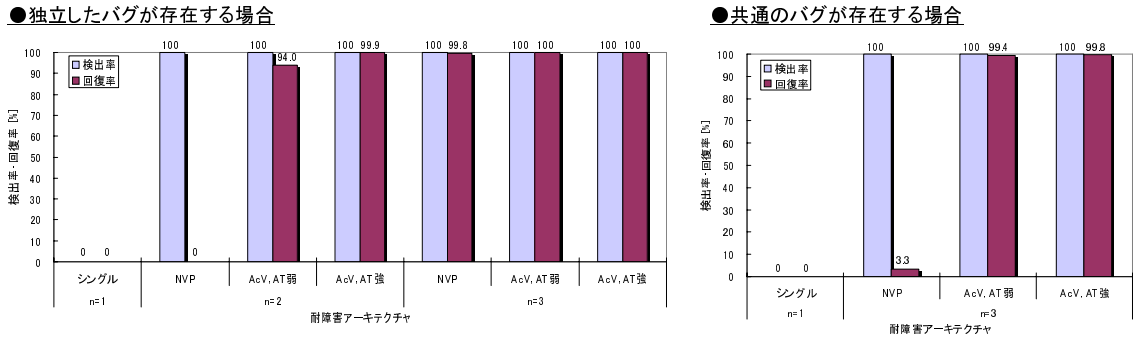

図 2 独立したバグが存在する場合（左），および共通のバグが存在する場合（右）の耐障害性（単位：%）
Fig. 2 The fault tolerance with independent(left) and dependent(right) bugs

表 4 共通のバグが存在する場合の正確性（単位：%）
Table 4 The accuracy with dependent bugs

耐障害アーキテクチャ		正解率	誤り率	例外率
n=1	シングル	93.19	6.81	0
n=3	NVP	89.96	10.02	0.02
	AcV, AT 弱	99.94	0.04	0.02
	AcV, AT 強	99.98	0	0.02

と AcV では，中間出力の異常を検出でき，最終出力が異常値となることを低減できるためである．本評価で用いたバグ A〜C の場合は，バグのすべての顕在化を検出でき，誤り率をゼロとすることができている．

例外率に関しては，n=2 と n=3 の双方の項目において，AcV の方が NVP よりも減少している．この理由としては，AcV が，中間出力の異常を検出した場合に回復できる割合が高いのに対し，NVP は，異常を検出したものの回復できない場合があるために，例外を発行する割合が高まるためである．特に，n=2 の NVP は耐障害アーキテクチャの特性上，回復を行うことができない．AcV の正解率の高さが誤り率と例外率の低さにも表れているといえる．

(2) 共通のバグが存在する場合

共通のバグが存在する場合（バグインジェクション数 2）の正確性を**表 4** に示す．(1) と同様，バリアント数 n=1 の場合は，表 1 に示す 3 パターンの平均値を示す．この結果について，以下のように考察できる．

[AcV で正解率が向上]

n=3 の NVP では，正解率が 89.96%であるのに対し，n=3 の AcV は，99.94%〜99.98%と NVP を上回る正解率が得られている．n=3 の AcV を用いることで，2 個のバリアントにおいて同時にバグが顕在化した場合でも，受理テストによって，それら 2 個のバリアントの中間出力を異常と判定し，残りの 1 個の中間出力を最終出力として採用できるためである．一方，n=3 の NVP は，シングルよりも正解率が低下し，かつ誤り率が増加している．この理由は，2 個のバリアントにおいて同時にバグが顕在化した場合に，2 個の等しい異常値が中間出力として出力されるために，その異常値を決定器の多数決で最終出力として採用してしまうからである．共通のバグが含まれている場合に，AcV が NVP よりも高い正確性が得られる一例が示されているといえる．

[AcV で受理テストが厳密なほど誤り率が低減]

AcV の AT 弱と AT 強の誤り率を比較してみると，それぞれ 0.04%とゼロで，AT 強の方が誤り率を低減できている．その理由としては，受理テストが厳密であればあるほど，バリアントのバグの顕在化を検出でき，異常な中間出力が決定器に入力される割合が低減されるため，結果として誤り率が低減されるといえる．厳密な受理テストが誤り率を低減させることがわかる．

4.2 耐障害性の評価

(1) 独立したバグが存在する場合

独立したバグが存在する場合（バグインジェクション数 1）の耐障害性を**図 2**（左）に示す．本図では，表 2 に示した耐障害性の評価項目のうち，検出率と回復率を示している．正確性の評価と同様，n=1 と n=2 の場合は，表 1 に示す 3 パターンの平均値を用いる．

検出率については，NVP と AcV を適用した場合，すべての項目で 100%となっている．これは，NVP では，インジェクションしたバグが顕在化したときに，必ず各バリアントの中間出力が異なったことを表している．同様に AcV では，各バリアントの受理テストのうちどれかでパスしないケースが発生したか，あるいは受理テストで正しいと判定された中間出力が各バリアント間で異なったことを表している．NVP，および AcV は，バグの顕在化の検出に有効といえる．

一方，回復率に関しては，以下のように考察できる．

[**AcV で受理テストが厳密なほど回復率が向上**]

n=2 の AcV において AT 弱と AT 強の回復率はそれぞれ 94.0%と 99.9%となっており，AT 強の方が回復率が向上している．この理由としては，受理テストが厳密であればあるほど，中間出力の異常をもれなく判断できるため，決定器に入力される中間出力が真に正常である割合が高まり，それゆえ最終出力として正常な値を出力できるからといえる．逆に言えば，受理テストに厳密さを欠くほど，すなわち正常であることの必要条件を緩くすればするほど，決定器に入力される中間出力が異常である割合が高まり，最終出力として異常値を出力することにつながるといえる．

[**AcV で回復率が向上**]

n=2 の AcV（AT 強）と n=3 の NVP の回復率を比較してみると，AcV（AT 強）の方が高い結果となっている．この理由としては，n=3 の NVP においては，異常が発生したバリアントの中間出力の個数が，正常なバリアントの中間出力の個数を上回ったために，異常を検出しながらも回復までに至らない場合があったのに対し，n=2 の AcV では，厳密な受理テストによって，正常なバリアントの中間出力を最終出力に採用できたためである．NVP のバリアント数の増加よりも，AcV の厳密な受理テストの方が，回復率の向上をもたらす場合があるといえる．

（2）共通のバグが存在する場合

共通のバグが存在する場合（バグインジェクション数 2）の耐障害性を図 2（右）に示す．本図について，以下のように考察できる．

[**NVP で回復率が低下**]

n=3 の NVP においては，回復率が 3.3%となっている．表 1 で示したように，n=3 でバグインジェクション数が 2 の場合は，バグ A+B-バグ A+C-バグ B+C のように，3 個のうち 2 個のバリアントに同一のバグがインジェクションされる．したがって，2 個のバリアントで同時にバグが顕在化する割合が高まり，その際には NVP で回復ができずに例外が発生する．回復率が低いのはそのためである．

[**AcV で回復率が向上**]

n=3 の AcV の場合は，回復率が，AT 弱と AT 強それぞれ 99.4%と 99.8%となっており，NVP の回復率を大幅に上回る結果が得られている．これは，AcV が，NVP の場合とは異なり，2 個のバリアントで同時にバグが顕在化したとしても受理テストで中間出力の異常を検出でき，残り 1 個のバリアントの正常な中間出力を最終出力に採用できるからである．

4.3 まとめと考察

以上の評価結果より，車載通信層に AcV を適用した場合は，シングル，ならびに NVP を適用した場合と比較して，正確性（正解率）と耐障害性（回復率）が向上することがわかり，結果として信頼性が向上することが定量的に示されたといえる．本評価で得られた知見は，以下の 3 点にまとめられる．すなわち，(1) バリアント数 n=2 の AcV が n=3 の NVP よりも高い正確性が得られる場合があり，NVP のバリアント数の増加よりも，AcV の厳密な受理テストの方が，回復率の向上をもたらす可能性が示されたこと，(2) AcV の受理テストが厳密なほど誤り率が低減し，正解率と回復率が向上すること，および（3）共通のバグが存在する場合において，AcV の方が，NVP を大幅に上回る正解率と回復率を得ることができることである．

これらの知見よりわかることは，AcV が有効に機能するうえで重要な点は，実行時の動作が仕様を満たしているかどうかをチェックできる厳密な受理テストが必要不可欠なことである．本稿における AcV の適用例である車載通信層の定期送信に関しては，「オフセットを伴って一定周期でメッセージが CAN レジスタに書き込まれる」という仕様が満たされているかどうかを実行時にチェックする受理テストがアサーションで厳密に作成できる．よって，AcV が有効に機能するがゆえに，NVP よりも AcV の方が正確性と耐障害性が向上する結果が得られていると考察する．逆に，非常に複雑な演算を行う車載制御アプリケーションであるために，入出力と内部状態を用いて厳密な受理テストを作成するのが困難な場合，AcV は，NVP よりもわずかに優位である程度にとどまると考える．なぜならば，この場合の AcV では，厳密さを緩めた受理テストにせざるを得ず，厳密な受理テストの場合と比較して回復率が得られないからである．したがって，受理テストを厳密に作成できるアプリケーションであるか否かが，AcV の有効性に大きく関わるといえる．

さらに留意すべき点は，以上の評価が，特定の 3 種類のバグとそれらを顕在化する 1 つのバグ顕在化シナリオにおける結果であることである．一層多くの種類のバグとバグ顕在化シナリオで評価を行うことによって，評価結果の精度が高まり，さらには，本来満たすべき正確性と耐障害性の目標値をクリアしているか否かの判断をより正確に行えるといえる．

5. おわりに

実際の組込みソフトウェアに対して NVP 以外の耐障害アーキテクチャを適用することでどれほどの信頼

性が得られるのかが定量的に明らかになっていないという背景のもと，本研究では，車載通信層を例としてAcV を適用し，正確性（正解率）と耐障害性（回復率）で信頼性を定量化して，バグインジェクションにより信頼性の定量的な評価を行った．その結果，車載通信層に AcV を適用した場合，シングル，ならびにNVP を適用した場合と比較して，正確性と耐障害性が向上することがわかり，信頼性が向上することが定量的に示されたといえる．

今後の課題としては，インジェクションするバグの種類，およびバグ顕在化シナリオにバリエーションを持たせた場合の信頼性評価，信頼性を一層向上させる新たな耐障害アーキテクチャの検討等が挙げられる．

参 考 文 献

1) Cygwin project. http://www.cygwin.com/.
2) ISO/IEC 9126-3. Software engineering - product quality - part3: Internal metrics, 2003.
3) X. Cai, M. Ryu, and M. Vouk. An experimental evaluation on reliability features of N-version programming. In *proc. of the 16th IEEE Int'l Symp. on Software Reliability Engineering(ISSRE '05)*, pp. 161–170, 2005.
4) J. Duraes and H. Madeira. Emulation of software faults - a field data study and a practical approach. *IEEE Transactions on Software Engineering*, Vol. 32, No. 11, pp. 849–867, 2006.
5) ISO26262-6. Road vehicles - functional safety - part6: Product development at the software level, 2011.
6) C. Kaner, J. Falk, and H. Nguyen. *Testing Computer Software*. Wiley, 1999.
7) W. Ng and P. Chen. Systematic improvement of fault tolerance in the rio file cache. In *proc. of IEEE Fault Tolerant Computing Symp.*, pp. 76–83, 1999.
8) L. Pullum. *Software Fault Tolerance Technics and Implementation*. Artech House Publishers, 2001.
9) P. Townend, J. XU, and M. Munro. Building dependable software for critical applications: Multi-version software versus one good version. In *proc. of the 6th Int'l Workshop on Object-Oriented Real-Time Dependable Systems(WORDS '01)*, pp. 103–110, 2001.
10) T. Wilfredo. Software fault tolerance: a tutorial. Technical report, NASA, 2000.
11) クリアコード. Cutter. http://www.clear-code.com/software/cutter.html.
12) 経済産業省. 2010 年版組込みソフトウェア産業実態調査報告書, 2010.